Homework 6: Q3

Name: Waiwai Kim, waiwaiki

1 Part (a): Proof Idea

We are asked to argue that there is always an optimal solution with no idle time by induction that $foreveryi, c'_i \leq c_i$. The proof idea is pretty simple, as we have seen in classes. If there is an idle time between two consecutive schedules, we can simply 'squeeze' them together, and the resulting finish time of the new schedule is not worse than the finish time of the original schedule. We can run the 'squeeze' operation for every pair of schedule from start to finish. If there is no idle time in a pair, we can do nothing.

If we find a case where an idle time exists, as $st_{i+1} > st_i + s_i$ indicates, we can squeeze i and $i+1$. The idle time is $idle_{i+1} = diff(st_{i+1}, st_i + s_i)$. The new starting time $st'_{i+1} = st_i + s_i = st_{i+1} - idle_{i+1}$. Because $st'_{i+1} < st_{i+1}$, we can say that the new completion time c'_i is less than c_i where $c_i = st_i + s_i + b_i + r_i$. If we find a case where an idle time does not exist, we can say that $st'_{i+1} = st_{i+1}$ and $c'_i = c_i$. We know that every single case either has an idle time or doesn't have idle time. Thus, if we have a schedule that is not optimal, we can perform the aforementioned operation for every single pair of $(i, i+1)$. Accordingly, we can say that for every single schedule, $st'_i \leq st_i$ and $c'_i \leq c_i$. Here we have converted not optimal solution to an optimal solution.

2 Part (b): Algorithm Idea

Let's assume that we have total n campers from $1...n$. For notation, we can say for a given $camper_i$, he or she has swimming time of s_i, biking time of b_i and running time of r_i.

The algorithm idea is to line up campers in decreasing order of $b_i + r_i$. Said differently, the camper who will take the longest time to finish biking and running irrespective of swimming will start the first. The camper who is the fastest cyclist and runner combined will be the last person to start.

3 Part (b): Algorithm Details

Let's assume the input to the algorithm is a list of tuple where the second element is person number, the second element is swimming time, the third element is biking time and the fourth element is the running time. For example for $camper_i$ has (i, s_i, b_i, r_i).

First, in line 2 to 5 of Algorithm 1, we reconstruct a list by summing up b_i and r_i. We call this new_list. In line 6, we will sort it by a descending order. Note that $reverse = True$ in Python allows us to do so. In line 7, the start time for the first element is 0. In line 8, we construct a list of finish time for each camper so that we can know when the competition will end.

In line 9, we iterate every single camper in the new_list. In line 10, the finish time for a given $camper_i$ is $starr_i + s_i + (b_i + r_i)$ where $s_i == new\_list[i][1]$ and $b_i + r_i == new\_list[i][2]$. In line 11, we set the starting time for the next camper.

In line 13-14, we figure out the maximum finish time for all campers and print the result. In the end, we return the new_list. Our friend can schedule according to the first element for every tuple in new_list because this element is the camper number in a decreasing order of sum of biking and running time.

Algorithm 1 Algorithm1

```
procedure ALGORIGHM-1(Inputs)
    new_list = []
    for camper in Inputs: do
        new_list.append((camper[0],camper[1], camper[2]+camper[3]))
    end for
    new_list.sort(key=operator.itemgetter(1), reverse = True)
    start = 0
    finish_list =[]
    for  i in range(len(new_list)) : do
        finish_list.append( start + new_list[i][1] + new_list[i][2])
        start += new_list[i][1]
    end for
    max_finish_time = max(finish_list)
    print("the earliest finish time is ", max_finish_time)
     return new_list
end procedure
```

4 Part (b): Proof of Correctness Idea

As we have seen in class to minimize the maximum lateness, we can use an exchange argument to prove the correctness of the algorithm. Here, we will modify the idea of inversion. Here, an inversion exist when j starts earlier than k when $b_j + r_j < b_k + r_k$. We can argue that when we swap j and k to fix the inversion, the finish time of the swapped schedules is earlier than the inversed schedule. Note that this removes 1 inversion. As we have seen in class we can do so i times from O' where $idle(O') = 0$ and $\#inv(O') > 0$ to arrive O where $idle(O) = 0$ and $\#inv(O) = 0$.

5 Part (b): Proof Details

Please refer to October 24th lecture notes regarding how this procedure has properties of $idle(O_i) = 0 \implies idle(O_{i+1}) = 0$ and $\#inv(O_i) > 0 \implies \#inv(O_{i+1}) = \#inv(O_i) - 1$. Here we will focus on the property that states that the finish time of O_{i+1} is less than or equal to O_i or $f(O_{i+1}) \leq f(O_i)$ by an exchange argument.

Consider a solution where an inversion exists. j starts earlier than k when $b_j + r_j < b_k + r_k$. (j,k) is an inversion. Let's swap (j,k) to (k,j). In the case of inversion, the second person or camper$_k$ starts biking at $s_j + s_k$. Similarly, in the case of non-inversion or (k,j), the second person or camper$_j$ starts biking at $s_k + s_j$. Note that in both cases, the second person starts biking at the same time of $s_j + s_k$. Let's call $s_j + s_k$ as t_0. In order to tell which schedule ends earlier, we only have to know which person takes longer to finish biking and running because the starting time for the second person biking and running is the same.

Said differently, in (j,k) or O_i where an inversion exists, the finish time is $t_0 + b_k + r_k$. On the other hand, in (k,j) or O_{i+1}, the finish time is $t_0 + b_j + r_j$. We know that $(t_0 + b_k + r_k) > (t_0 + b_j + r_j)$ because $b_k + r_k > b_j + r_j$. Thus our swapped schedule has a finish time no greater than the unswapped schedule.

We can continue to do so in order to eliminate all inversions without increasing finish time. At the end of the process, we will have a schedule that is produced by our algorithm whose finish time is not greater than the optimal solution. Thus the algorithm presented here must be optimal.

6 Part (b): Runtime Analysis

The lines 3-4 of Algorithm 1 takes $O(n)$ because every single camper is iterated. This assumes appending in line 4 takes $O(1)$. The sorting algorithm on an array in line 6 will take $O(nlogn)$. The for loop in line 9 to 12 will take $O(n)$ assuming the appending and summation will take $O(1)$. Finding the maximum in line 13 takes $O(n)$. Printing in line 14 takes $O(1)$. The overall runtime of the algorithm is $O(n)+O(nlogn)+O(n)+O(n) = O(n+nlogn+n+n) = O(nlogn)$.